第九章 人工神经网络

安徽理工大学计算机学院 方贤进
（春季，研究生课程）

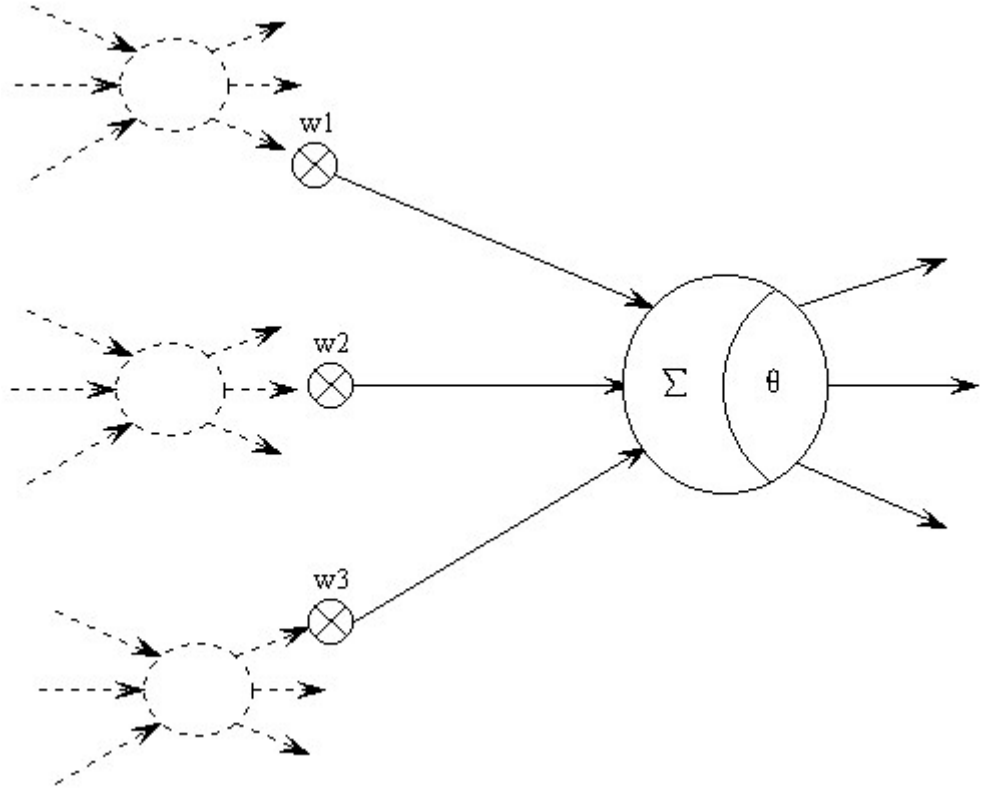
神经网络是通过模拟人脑的结构和工作模式，使机器具有类似人类的智能，如机器学习、知识获取、专家系统等。

9.1 人工神经网络的概念、结构

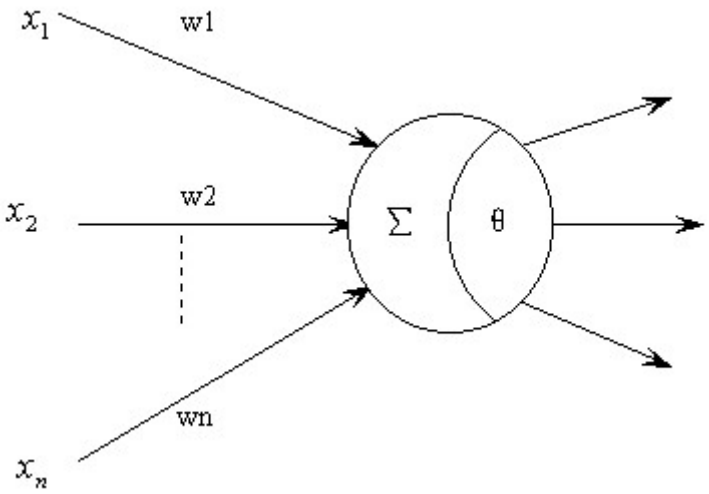
人工神经网络（Artificial Neural Nets，ANN）是由大量处理单元经广泛互连而组成的人工网络，用来模拟脑神经系统的结构和功能。而这些处理单元称作人工神经元。

人工神经网络（ANN）可以看成是以人工神经元为结点，用有向加权弧连接起来的有向图。在此有向图中，人工神经元就是对生物神经元的模拟，而有向弧则是轴突—突触—树突对的模拟。有向弧的权值表示相互连接的两个人工神经元间相互作用的强弱。

人工神经网络的组成：



人工神经元模型由心理学家 Mcculloch 和数理逻辑学家 Pitts 合作提出的 M-P 模型：



$x_1, x_2, \ldots, x_n$是来自其它人工神经元的信息作为该人工神经元的输入，权值 $w_1, w_2, \ldots, w_n$表示各输入的连接强度。 θ 是神经元兴奋时的内部阈值，当神经元输入的加权和大于 θ 时，神经元处于兴奋状态。而神经元的输出为：

$$y = f\left(\sum_{i=0}^{n-1} w_i x_i - \theta\right)$$

，f称为激发函数或作用函数，该输出为 1 或 0 取决于其输入之和大于或小于内部阈值。。也就是说令：

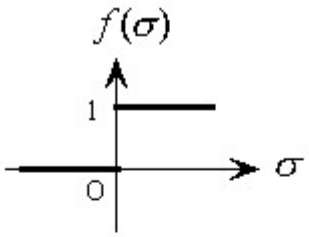
$$\sigma = \sum_{i=0}^{n-1} w_i x_i - \theta$$

，f 函数的定义如下：

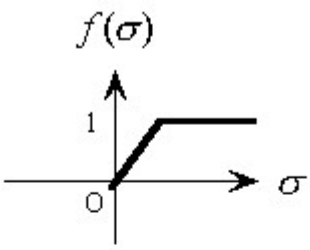
$$y = f(\sigma) = \begin{cases} 1, \sigma > 0 \\ 0, \sigma < 0 \end{cases}$$

即 σ>0 时，该神经元被激活，进入兴奋状态，f(σ)=1,当 σ<0 时，该神经元被抑制，f(σ)=0

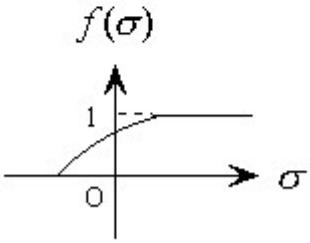
激发函数具有非线性特性。常用的非线性激发函数有阈值型、分段线性型、Sigmoid 函数型（简称 S 型）和双曲正切型。如下图所示：



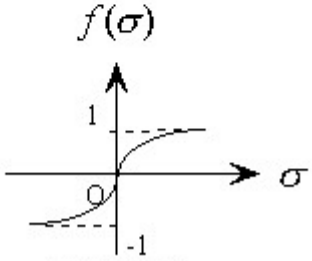
阈值型



分段线性



Sigmoid函数型



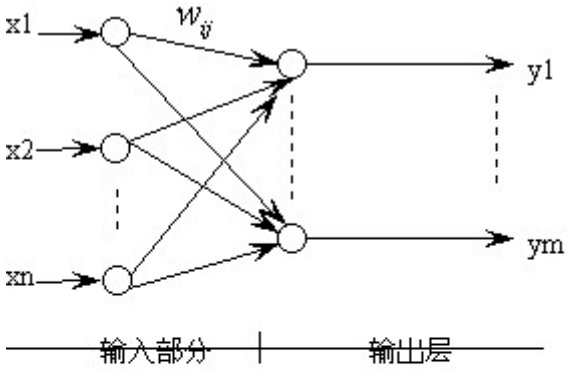
双曲正切型

9.2 单层感知器（Perceptron）模型及其学习算法

9.2.1 感知器训练法则

1957 年美国学者Rosenblatt提出了一类具有自学习能力的感知器模型，它是一个具有单层计算单元的前向神经网络，其神经元为线性阈值单元，称为单层感知器。它和M-P模型相似，当输入信息的加权和大于或等于阈值时，输出为 1，否则输出为 0 或-1。与M-P模型不同之处是神经元之间的连接权值w_i是可变的，这种可变性就保证了感知器具有学习能力。

Rosenblatt 提出的单层感知器由输入部分和输出层构成，输出层即为它的计算层。在该感知器模型中，输入部分和输出层都可由多个神经元（处理单元）构成，输入部分的神经元与输出层的各种神经元间均有连接。当输入部分将输入数据传送给连接的处理单元时，输出层就会对所有输入数据进行加权求和，经阈值型作用函数产生一组输出数据。



$$y_i = \begin{cases} 1, & 若\sum_{i=1}^{n} w_{ij}x_i - \theta_j \geq 0 \\ 0, & 若\sum_{i=1}^{n} w_{ij}x_i - \theta_j < 0 \end{cases}$$

$$j = 1,2,\ldots,m$$

1959 年 Rosenblatt 提出了感知器模型中连接权值参数的学习算法。算法的思想是首先把连接权值和阈值初始化为较小的非零随机数，然后把有 n 个连接权值的输入送入网络，经加权运算处理，得到的输出如果与所期望的输出有较大的差别，就对连接权值参数按照某种算法进行自动调整，经过多次反复，直到所得到的输出与所期望的输出间的差别满足要求为止。

感知器信息处理的规则为：

$$y(t) = f\left[\sum_{i=1}^{n} W_i(t)x_i - \theta\right],$$

其中 y(t)为 t 时刻输出，xi 为输入向量的一个分量，Wi(t)为 t 时刻第 i 个输入的加权， θ 为阈值，f()为阶跃函数。

感知器的学习规则如下：

$$W_i(t+1) = W_i(t) + \eta[d - y(t)]x_i$$

其中 η 为学习率（0<η<1），d 为期望输出，y(t)为实际输出。

单层感知器学习算法(learning algorithm for single layer perceptron):

step1:initialize connection weight and threshold. set a smaller nonzero random value for *wi*(i=1,...,n) and θ as their initial value. wi(0) represents connection weight of *i*-th input at the moment of *t*;

step2:input a training parameter *X=(x1(t),x2(t),...,xn(t))*and the expectation output *d(t)*;

step3:compute the actual output of ANN:

$$y(t)=f(\sum_{i=1}^{n}\omega_i(t)x_i(t)-\theta),i=1,2,\cdots,n$$

step4:compute the difference value between actual output and expectation output:

DEL=d(t)-y(t)

if DEL< ε (ε is a very small positive number), then training of ANN is over, otherwise goto step 5;

step5: regulate the connection weight according to the following fomular:

$$\omega_i(t+1)=\omega_i(t)+\eta(d(t)-y(t))x_i(t),i=1\cdots n$$

where 0< η <=1 is an incremental factor, it is used to control regulation speed, and called learning rate. Usualy, the value of η can't be too large or small. if its value is too large, then η will impact the convergence of wi(t),otherwise make the convergence speed of wi(t) slower;

step 6: goto step 2;

9.2.2 线性不可分问题

1962 年 Rosenblatt 宣布人工 神经网络可以学会它能表示的任何东西。但是 1969Minsky 发表了一书《perceptron》，书中指出单层感知器不能解决许多最基本的问题，如异或问题（XOR），这类问题统称为线性不可分问题，即输入的训练样本集是否是线性可分的，如果是线性可分的，则训练算法收敛，否则不收敛。

异或问题的定义如下：

$$y(x_1,x_2)=\begin{cases}0, \ if \ \ x1=x2\\ 1, 其它\end{cases} \qquad (9.1)$$

相应的真值表如下：

点	输入 x1	输入 x2	输出 y
A1	0	0	0
B1	1	0	1
A2	1	1	0
B2	0	1	1

如果用单层感知器的话，其输出为

$$y(x1,x2)=f(\omega1*x1+\omega2*x2-\theta) \qquad (9.2)$$

如果用单层感知器解决异或问题的话，根据异或问题的定义以及真值表可知，θ ω1， ω1 的取值必须满足下面方程组：

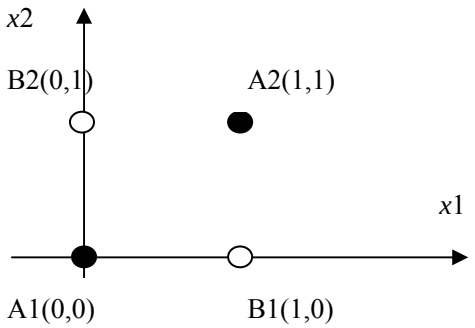
$$\begin{cases}0+0-\theta<0\\ \omega1-\theta\geq0\\ \omega1+\omega2-\theta<0\\ \omega2-\theta\geq0\end{cases} \qquad (9.3)$$

该方程组是无解的。所以单层感知器是无法解决异或问题的。

该问题的几何意义如图所示，感知器的输出为 1 用空心圆表示，输出为 0 用实心圆表示，可以看出满足 x1 XOR x2=1 的点集（输入集）为 B={B1，B2}，满足 x1 XOR x2=0 的点集（输入集）为 A={A1，A2}。显然无论如何选择 θ， ω1， ω1 的取值，

都无法找到一条直线在该平面上将 A、B 两类输入集分开，即使其它的激发函数也很难做到。这种单层感知器所不能表达的问题称为线性不可分问题。

事实上很多问题都不能用单层感知器表达，这也是单层感知器的缺陷。



9.2.3 梯度下降和 Delta 法则

尽管训练样本线性可分时，感知器法则可以成功地找到一个权向量，但如果样本不是线性可分的它将不能收敛。因此人们设计了另一个训练法则来克服这个不足，称为 Delta 法则（delta rule）。如果训练样本不是线性可分的，那么 delta 法则会收敛到目标概念的最佳近似。

Delta 法则的关键思想是使用梯度下降（gradient descent）来搜索可能的权向量的假设空间，以找到最佳拟合训练样本的权向量。

可以把 Delta 训练法则理解为训练一个无阈值的感知器，也就是一个线性单元(Linear unit)，它的输出 o 如下：

$$o(\vec{x}) = \vec{\omega} \bullet \vec{x} \qquad (9.4)$$

为了推导线性单元的权值学习法则，先指定一个度量标准来衡量假设的权向量相对于训练样本的训练误差（training error）：

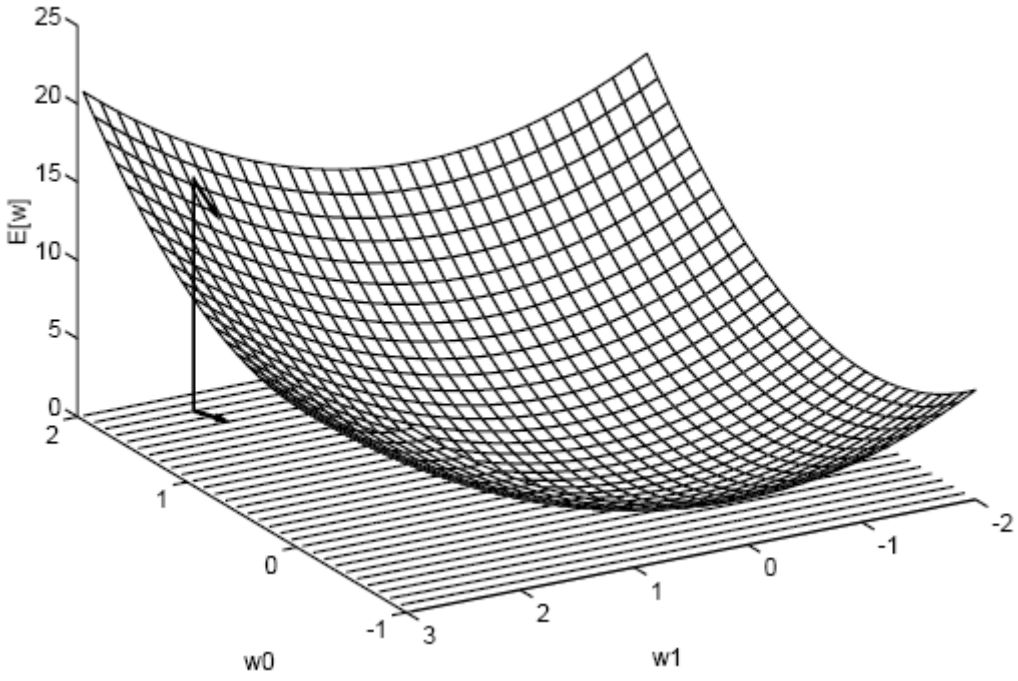
$$E(\vec{\omega}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 \qquad (9.5)$$

其中 D 是训练样本集合，t_d 是训练样本 d 的目标输出，o_d 是线性单元对训练样本 d 的输出。在这里把 E 定义为 $\vec{\omega}$ 的函数，是因为线性单元的输出 o 依赖于这个

权值，当然 E 也依赖于特定的训练样本集合，但一般在训练期间训练样本集合是固定的，所以没有把 E 也定义成训练样本的函数。

1. 可视化假设空间

为了理解梯度下降算法，可视化地表示包含所有可能的权向量和相关联的 E 值的整个假设空间。如图所示。这里坐标 ω_0,ω_1 表示一个简单的线性单元中两个权可能的取值，纵轴表示相对于某固定训练样本的误差 E。因此图中的误差曲面概括了假设空间中每一个权向量的期望度（desirability）(期望得到一个具有最小误差的假设)。如果定义 E 的方法已知，那么对于线性单元，这个误差曲面必然是具有单一全局最小值的抛物面。当然抛物面的形状依赖于具体的训练样本集。



为了确定一个 E 最小化的权向量，梯度下降搜索从一个任意的初始权向量开始，然后以很小的步伐反复修改这个向量。每一步都沿误差曲面产生最陡峭下降的方面修改权向量，继续这个过程直到得到全局最小误差点。

2. 梯度下降法则的推导

如何计算出沿误差曲面最陡峭下降的方向？可通过计算 E 相对于向量 $\vec{\omega}$ 的每个分量的偏导数来得到这个方向。这个向量导数被称为 E 对于 $\vec{\omega}$ 的梯度，记作 $\nabla E(\vec{\omega})$

$$\nabla E(\vec{\omega}) \equiv \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \cdots, \frac{\partial E}{\partial \omega_n} \right] \qquad (9.6)$$

当梯度被解释为权空间的一个向量时，它确定了使 E 最陡峭上升的方向，那么其反方向就是最陡峭下降方向。上图中的箭头显示了 ω0, ω1 平面的一个特定点的负梯度$-\nabla E(\vec{\omega})$。那么梯度下降的训练法则是：

$\vec{\omega} \leftarrow \vec{\omega} + \Delta\vec{\omega}$，其中 $\Delta\vec{\omega} = -\eta \nabla E(\vec{\omega})$ (9.7)

η称为 learning rate,它决定梯度下降搜索中的步长。公式中的负号是因为我们想让权向量向 E 下降的方向移动。这个训练法则可以写成它的分量形式：

$\vec{\omega_i} \leftarrow \vec{\omega_i} + \Delta\omega_i$，其中 $\Delta\omega_i = -\eta \frac{\partial E}{\partial \omega_i}$ (9.8)

根据 9.5 和 9.8 可以计算权值迭代更新的算法，以下是推导过程。

$$\frac{\partial E}{\partial \omega_i} = \frac{\partial}{\partial \omega_i} \bullet \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial \omega_i} (t_d - o_d)^2 = \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial \omega_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial \omega_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial \omega_i} (t_d - \vec{\omega} \bullet \vec{x_d})$$

$$\frac{\partial E}{\partial \omega_i} = \sum_{d \in D} (t_d - o_d)(-x_{id}) \qquad (9.9)$$

其中$-x_{id}$表示训练样本 d 的一个输入分量x_i

将 9.9 代入 9.8 得到梯度下降权值更新法则：

$$\Delta\omega_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \qquad (9.10)$$

以下是训练线性单元的梯度下降算法：

Gradient-Descent(training_examples, η)

//training_examples 中每个训练样本形式为序偶$<\vec{x},t>$,其中$\vec{x}$是输入值向量，t 是目标输出值， η是学习速率

Step1: 初始化每个 wi 为某个小的随机值；

Step2: 遇到终止条件之前，做以下操作

Step2.1 初始化每个$\Delta w_i = 0$

Step2.2 对训练样本 training_examples 中的每个$<\vec{x},t>$，做

Step2.2.1 把样本实例$\vec{x}$输入到此单元，计算输出 *o*;

Step2.2.2 对于线性单元的每个权 wi，做

$$\Delta w_i = \Delta w_i + \eta(t-o)x_i ; \qquad (9.11)$$

Step2.3 对于线性单元的每个权 wi,做

$$w_i = w_i + \Delta w_i \qquad (9.12)$$

3. 梯度下降的随机近似

应用梯度下降的主要实践问题是：（1）有时收敛过程可能非常慢（需要数千步）；（2）如果在误差曲面上有多个局部极小值，那么不能保证会找到全局最小值。

解决这个问题的办法是一种改进，即增量梯度下降（incremental gradient descent）或随机梯度下降（stochastic gradient descent）。公式 9.10 给出的梯度下降法则在对 D 中所有训练样本求和后计算权值更新，随机梯度下降的思想是根据每个单独样本的误差增量计算权值更新，得到近似的梯度下降搜索。将公式 9.10 改成：

$$\Delta w_i = \eta(t-o)x_i \qquad (9.13)$$

对应地，将梯度下降算法修改只要删除其中的公式 9.12,另外再将 9.11 替换成公式 9.14 即可

$$w_i = w_i + \eta(t-o)x_i \qquad (9.14)$$

公式 9.14 中的训练法则被称为增量法则，或叫 LMS 法则（Least-Mean-Square, 最小均方）、Adaline 法则或 Windrow-Hoff 法则。

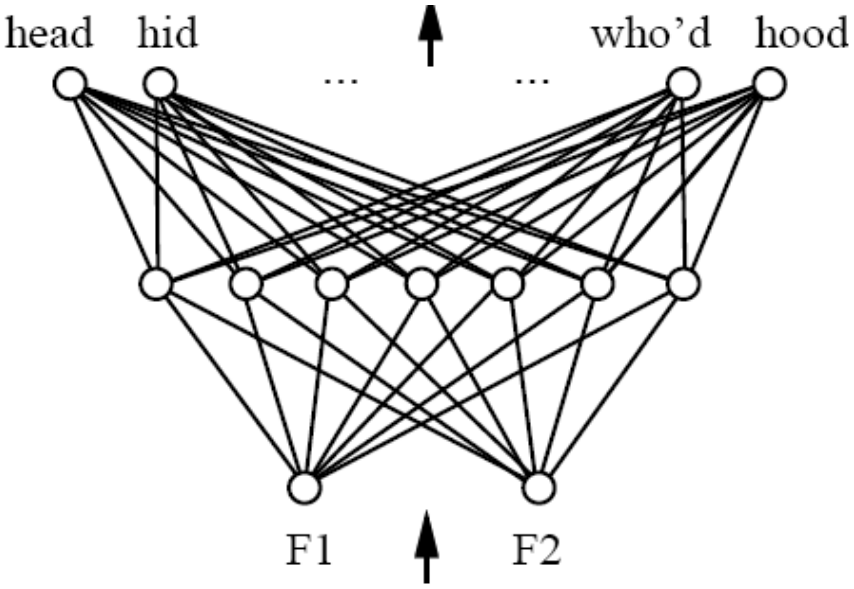
9.3 多层网络、反向传播模型及其学习算法

9．3．1 Context

- 单层 perceptron 只能表示线性决策面；
- 反向传播模型（Back-Propagation, 又称 B-P 模型）能够表示种类繁多的非线性曲面；

9.3.2 例子

- Example1: 一个多层网络表示一个语音识别任务

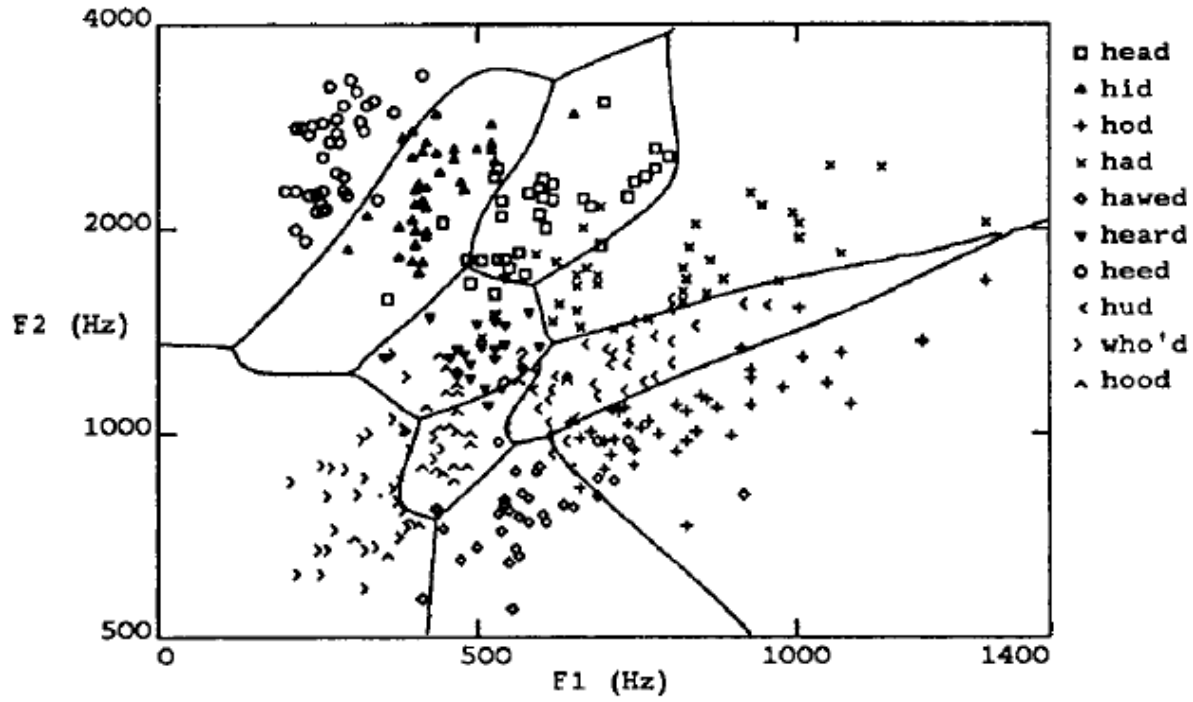


语音识别任务主要是区分出现在 h_d 中 10 种元音（e.g hid, had, head, hood, who'd 等）。输入的参数 F1，F2 是通过对声音的频谱

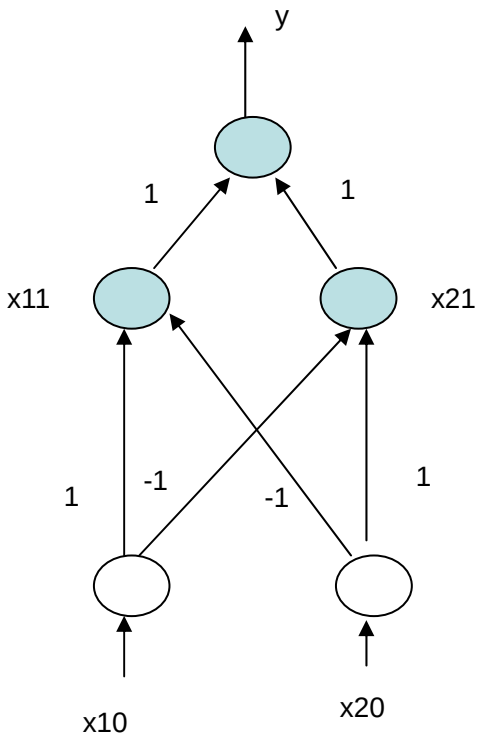
进行分析得到的，网络的 10 个输出对应于 10 个可能的元音，网络的预测是其中有最大值的输出。关于激发函数的选择可参见以下文献

Huang, W. Y. &Lippmann, R, P(1988). Neural net and traditional classifiers. In Anderson (Ed.), *Neural Information Processing systems*(pp.387-396

语音识别神经网络所学习到的高层决策面



Example2: 二层感知器可以解决 XOR 问题



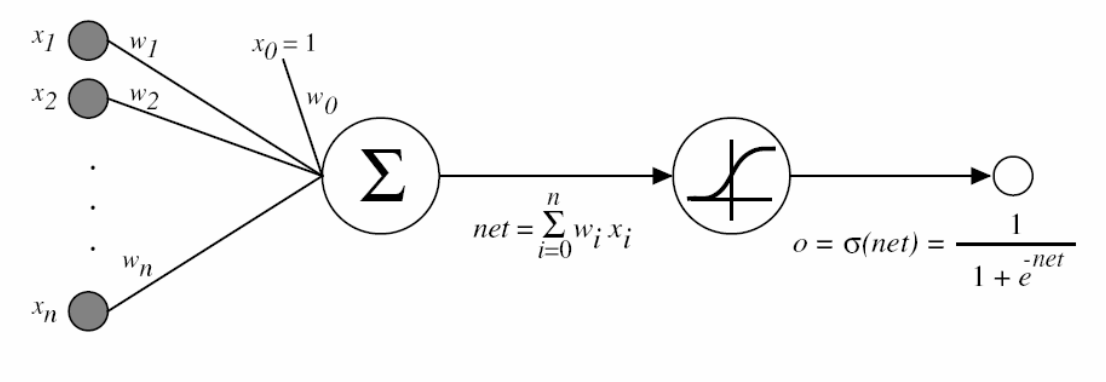
x11=1*x10+(-1)*x20-0.5

x21=1*x20+(-1)*x10-0.5

y=1*x11+1*x21-0.5

9.3.3 可微阈值函数

- 条件：
  - 输出是输入的非线性函数；
  - 输出是输入的可微函数；
  - 因此可以选择 sigmoid 函数



σ (x)是一个 sigmoid 函数，神经网络的输出

$o=\sigma(\vec{w}\bullet\vec{x})$，其中 $\sigma(y)=\frac{1}{1+e^{-y}}$

该函数的输出范围为（0，1]并且随输入单调递增。一个有用的特征是它的导数可以用它的输出表示

$$\frac{d\sigma(y)}{dy}=\sigma(y)\bullet(1-\sigma(y))$$

也可以用其它容易计算导数的可微函数来代替 σ，如：

$\sigma(y)=\frac{1}{1+e^{-k\bullet y}}$,其中$k$为某个正数来决定阈值函数的陡峭性

9．3．4 反向传播法则的推导

反向传播算法也是基于前面所讲的梯度下降法则的。首先定义一些符号：

- x_{ji}：单元 j 的第 i 个输入；
- w_{ji}：与单元 j 的第 i 个输入相关联的权值；

- $net_j = \sum_i w_{ji} x_{ji}$：单元 j 的输入的加权和；
- o_j：单元 j 计算出的输出；
- t_j:单元 j 的目标输出；
- σ：sigmoid 函数；
- Outputs:网络最后一层（输出层）的单元集合；
- Downstream(j):单元 j 的所有直接下游（immedialtely downstream）的单元的集合（也就是直接输入中包含单元 j 的输出的所有单元的集合）。

根据上节的内容，对于每个训练样本 d,每个权 w_{ji} 被增加 $\triangle w_{ji}$：

$$\triangle w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \qquad (9.15)$$

其中E_d是训练样本d的误差，是通过网络中所有输出单元的求和得到：

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

其中 t_k是单元k对于训练样本d的目标值，o_k是给定训练样本d时在单元k的输出值。

为了推导出$\frac{\partial E_d}{\partial w_{ji}}$的一个表达式，以便实现公式 9.15 的梯度下降算法的计算。首先注意权值w_{ji}仅能通过$net_j = \sum_i w_{ji} x_{ji}$影响网络的其它部分。所以可以根据偏导函数的链式规则得到：

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji} \qquad 9.16$$

接下来导出$\frac{\partial E_d}{\partial net_j}$的方便的表达式即可，分两种情况：

情况 1：单元 j 是输出层的一个单元，因而在该种情况下导出输出单元的权值训练法则。

就象w_{ji}仅能通过net_j影响网络一样，net_j仅能通过o_j影响网络。所以仍可通过链式规则得出：

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \qquad (9.17)$$

考虑 9.17 中的第一项

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

当 k<>j 时，所有输出单元 k 的导数$\frac{\partial}{\partial o_j}(t_k - o_k)^2$ 为 0，所以只求当 k=j 时：

$$\frac{\partial E_d}{\partial o_j}=\frac{\partial}{\partial o_j}\frac{1}{2}(t_j-o_j)^2=\frac{1}{2}\bullet 2\bullet(t_j-o_j)\frac{\partial(t_j-o_j)}{\partial o_j} \qquad (9.18)$$
$$=-(t_j-o_j)$$

接下来考虑 9.17 中的第二项。由于$o_j=\sigma(net_j)$，那么导数$\frac{\partial o_j}{\partial net_j}$就是 sigmoid 函数的导数，而 sigmoid 函数的导数为：

$\sigma(net_j)(1-\sigma(net_j))$，所以：

$$\frac{\partial o_j}{\partial net_j}=\frac{\partial\sigma(net_j)}{\partial net_j}=o_j(1-o_j) \qquad (9.19)$$

将公式 9.18 和 9.19 代入 9.17 中得到：

$$\frac{\partial E_d}{\partial net_j}=\frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}=-(t_j-o_j)o_j(1-o_j) \qquad (9.20)$$

根据 9.15，9.16 和 9.20 得到输出单元的梯度下降法则：

$$\triangle w_{ji}=-\eta\frac{\partial E_d}{\partial w_{ji}}=\eta(t_j-o_j)o_j(1-o_j)x_{ji}$$

情况 2：隐藏单元的权值训练法则

对于网络中的内部单元或者说隐藏单元的情况，推导w_{ji}必须考虑w_{ji}间接地影响网络输出，从而影响E_d，而net_j是通过

downstream(j)中的单元影响网络输出，并再影响E_d，所以可以进行如下推导：

先根据公式 9.20,设:

$$\delta_k = -\frac{\partial E_d}{\partial net_k} = -(-(t_k - o_k)o_k(1-o_k)) = (t_k - o_k)o_k(1-o_k)$$

那么有：

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j} \quad \text{(链式法则)} \\
&= \sum_{k \in downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in downstream(j)} -\delta_k w_{kj} o_j(1-o_j)
\end{aligned} \tag{9.21}$$

根据公式 9.15, 9.16 和 9.21 得到

$$\triangle w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$= -\eta x_{ji} \sum_{k \in downstream(j)} -\delta_k w_{kj} o_j (1 - o_j)$$

$$= \eta x_{ji} \sum_{k \in downstream(j)} \delta_k w_{kj} o_j (1 - o_j) \qquad (9.22)$$

$$= \eta x_{ji} o_j (1 - o_j) \sum_{k \in downstream(j)} \delta_k w_{kj}$$

如果用 $\delta_j = -\frac{\partial E_d}{\partial net_j}$，则得到明显的权值更新的反向传播方法：

$\delta_j = o_j(1-o_j) \sum_{k \in downstream(j)} \delta_k w_{kj}$ ，显然这是一个递归式

则权值更新的公式 9.22 可以写成：

$$\Delta w_{ji} = \eta x_{ji} \delta_j \qquad (9.23)$$

9．3．5 包含两层 sigmoid 单元的前馈网络的反向传播算法（随机梯度下降版本）

Backpropagation(training_example, η, n_{in}, n_{out}, n_{hidden})

//training_examples中每个训练样本是形如 $<\vec{x}, \vec{t}>$,其中 $\vec{x}$ 是网络输入值向量，$\vec{t}$ 是目标输出向量，η是学习速率，n_{in}是网络输入的数量，n_{out}是输出单元数量，n_{hidden}是隐含层单元数。

Step1: initialize all weights to small random numbers;

Step2: until satisfied, Do

Step 2.1 for each training example $<\vec{x},\vec{t}>$, Do

Step 2.1.1 input the training example to the network and compute the network outputs;

Step2.1.2 for each output unit k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

Step2.1.3 for each hidden unit h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$$

Step2.1.4 update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}, \text{ where } \Delta w_{ji} = \eta\delta_j x_{ji}$$

9.4 Hopfield 网络的工作原理

9.4.1 Hopfield 网络模型

BP 网络是一种前向多层网络，从输出层到输入层无反馈，缺少动态处理能力，计算能力不强。

Hopfield模型是Hopfield分别于1982及1984年提出的两个神经网络模型。1982年提出的是离散型的，1984年提出的是连续型的，但他们都是反馈网络结构，从输出层到输入层都有反馈存在。

在Hopfield网络模型中，网络的输出要反复地作为输入再送入网络中，这就使得网络具有了动态性。

9.4.1 Hopfield网络的学习算法

（1）设置互连权值

$$W_{ij}=\begin{cases}\sum_{s=0}^{m-1}x_i^s x_j^s, & i\neq j\\ 0, & i=j\end{cases}$$

其中x_s^i是S类洋例的第i个分量，它可以为1或0，样例类别数为m，结点数为n。

（2）未知类别样本初始化

$y_i(0)=x_i$ ，$0\leqslant i\leqslant n-1$

其中$y_i(t)$为结点i在t时刻的输出，当t=0时，$y_i(0)$就是结点i的初始值，x_i为输入样本的第i个分量。

（3）迭代直到收敛

$$y_j(t+1)=f\left(\sum_{i=0}^{n-1}W_{ij}y_i(t)\right),0\leq j\leq n-1$$

其中f为阈值型激发函数。该过程一直迭代到不再改变结点的输出为止，这时各结点的输出与输入样例达到最佳匹配。

（4）转（2）步继续

当Hopfield模型的网络中各神经元的连接权值所构成的矩阵是一个非负对角元的对称矩阵时或是一个非负定矩阵时，上述算法是收敛的。

9.5 用神经网络求解一个识别

手写识别试验

运行方法：

首先，在面板上写好字母点 input 键后，程序把面板上的图象信息转换为一个二进制向量，即按从左到右，从上到下的顺序检查面板，如果该位置被写过则对应 1，如果没被写过则对应 0。然后对该二进制向量进行单位化，单位化的目的是使得各种输入向量都有相等的长度，以便在第一层神经元中实现公平竞争。单位化了的向量送入神经网络进行计算。

运行原理：

PWriter2 所用的神经网络共两层，结构如图下图所示，注意图中大写粗体字母表示矩阵，小写粗体字母表示向量，小写细体字母表示标量。
网络的第一层有 256 个输入，26 个神经元，每个输入都和各个神经元相连，即每个神经元都能接收这 256 个输入。每个神经元有 256 个权值，是字母的标准写法，即每个神经元都储存着一个标准模式，26 个大写字母的标准写法分别由对应的这 26 个神经元储存。每个神经元的权值向量也是单位化的，同样是为了实现公平竞争。该层的传输函数为竞争函数。所谓竞争函数即每个神经元互相竞争，净输入最大者获胜，输出为 1，其余净输入较小的神经元竞争失败，输出为 0，以竞争函数为传输函数的神经元层每次只有净输入最大的那个神经元输出为 1，其余都为 0（不考虑净输入相等的情况）。

单神经元净输入的计算方法是输入向量的各个分量和权值向量的对应分量相乘，然后相加，再加上偏置值，即输入向量和权值向量的内积加上过竞争函数，该神经元输出 1，其余输出 0，这样一来就得到了一个 26 维向量，该向量进入第二层神经元继续计算。

第二层有 26 个输入，8 个神经元，26 个字母的 ASCII 码由这 8 个神经元的权值共同记录着，即每个神经元权值向量的第一个分量记录“A”的 ASCII 码，以二进制表示，第二个分量记录“B”……这层的传输函数为 hardlim 函数，该函数当净输入大于等于 0 时输出 1，小于 0 时输出 0。先前得到的向量进入该层后，由于只有一个分量为 1，所以每个神经元只有该位置的数据为原来的数，其余分量位置上的数据都为 0。再加上-0.5 的偏置，使原来
为 1 的量为 0.5，原来为 0 的量为-0.5，通过 hardlim 函数后所得的向量就为对应字母的 ASCII 码，再通过程序转换后就可输出该字母。

这个程序的问题还很多。由于第一层神经元记录的标准模式连字母在面板中的位置也记录了，所以即使当输入已经非常象一个标准模式，但位置偏了，所得的向量仍与标准向量差别较大，所以仍会认不准。而且，由于使用了竞争函数，所以即使人看来输入不象其中任何一个字母时，还是总会有一个标准向量与输入的内积最大，所以还是会认定一个错误的字母。

下载源代码